



Università  
degli Studi  
di Palermo



# Introduzione al linguaggio C

CALCOLATORI ELETTRONICI – FONDAMENTI DI PROGRAMMAZIONE  
a.a. 2024/2025

Prof. Roberto Pirrone



# Un programma per convertire le miglia in chilometri

```
/*  
 * Converts distance in miles to kilometers.  
 */  
#include <stdio.h> /* printf, scanf definitions */  
#define KMS_PER_MILE 1.609 /* conversion constant */  
  
int  
main(void)  
{  
    double miles, /* input - distance in miles. */  
    kms; /* output - distance in kilometers */  
  
    /* Get the distance in miles. */  
    printf("Enter the distance in miles> ");  
    scanf("%lf", &miles);  
  
    /* Convert the distance to kilometers. */  
    kms = KMS_PER_MILE * miles;  
  
    /* Display the distance in kilometers. */  
    printf("That equals %f kilometers.\n", kms);  
  
    return (0);  
}
```

commenti



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa



# Commenti

- Iniziano con `/*` e terminano con `*/`
- Il compilatore *ignora* il testo dei commenti
- Possono essere multilinea

# Un programma per convertire le miglia in chilometri

direttive di  
preprocessore

```
/*  
 * Converts distance in miles to kilometers.  
 */  
#include <stdio.h> /* printf, scanf definitions */  
#define KMS_PER_MILE 1.609 /* conversion constant */
```

commenti

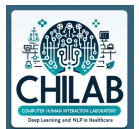
```
int  
main(void)  
{  
    double miles, /* input - distance in miles. */  
    kms; /* output - distance in kilometers */  
  
    /* Get the distance in miles. */  
    printf("Enter the distance in miles> ");  
    scanf("%lf", &miles);  
  
    /* Convert the distance to kilometers. */  
    kms = KMS_PER_MILE * miles;  
  
    /* Display the distance in kilometers. */  
    printf("That equals %f kilometers.\n", kms);  
  
    return (0);  
}
```



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa



# Direttive di preprocessore

- *preprocessore C*
- È un software che ha lo scopo di *modificare* parti del codice sorgente *prima* che inizi la compilazione
- Tutte le direttive iniziano con #

# Direttive di preprocessore

#define

- Sostituisce *value* a *NAME* in tutto il programma
- *value* è imm modificabile ed è una stringa sostitutiva che può essere complessa e che chiameremo *macro*
- Se la macro è semplicemente un valore, definiamo una costante

```
#define NAME value
```

```
#define MILES_PER_KM 0.62137
```

```
#define PI 3.141593
```

```
#define MAX_LENGTH 100
```

# Direttive di preprocessore

#include

- Include nel testo del programma gli *header file standard*

```
#include <standard header file>

#include <stdio.h>
#include <math.h>
```

- Contengono le definizioni delle funzioni della libreria standard
  - *Sono le API di sistema!!*
- Hanno estensione .h

# Un programma per convertire le miglia in chilometri

direttive di  
preprocessore

```
/*  
 * Converts distance in miles to kilometers.  
 */  
#include <stdio.h> /* printf, scanf definitions */  
#define KMS_PER_MILE 1.609 /* conversion constant */
```

commenti

```
int  
main(void)  
{  
    double miles, /* input - distance in miles. */  
    kms; /* output - distance in kilometers */  
  
    /* Get the distance in miles. */  
    printf("Enter the distance in miles> ");  
    scanf("%lf", &miles);  
  
    /* Convert the distance to kilometers. */  
    kms = KMS_PER_MILE * miles;  
  
    /* Display the distance in kilometers. */  
    printf("That equals %f kilometers.\n", kms);  
  
    return (0);  
}
```

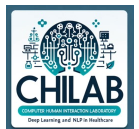
programma  
principale



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa





# Il programma principale

- La *funzione* `main` determina l'inizio dell'esecuzione
- `<tipo ritornato> <nome> (<lista parametri>)`
- `main` restituisce un valore intero al S.O.
  - Usa `return` per farlo
  - 0 vuol dire tutto ok
- `void` significa che non ci sono dati in ingresso

```
int
main(void)
{
    function body
}
```

```
int
main(void)
{
    printf("Hello world\n");
    return (0);
}
```

# L'istruzione `return`

- Un programma C può essere *modularizzato* usando più di una funzione
- Le funzioni *possono o meno* restituire un risultato
- `return` è l'istruzione che passa il risultato dall'interno della funzione che lo ha calcolato alla funzione chiamante

# Un programma per convertire le miglia in chilometri

```
/*  
 * Converts distance in miles to kilometers.  
 */  
#include <stdio.h> /* printf, scanf definitions */  
#define KMS_PER_MILE 1.609 /* conversion constant */  
  
int  
main(void)  
{  
    double miles, /* input - distance in miles. */  
    kms; /* output - distance in kilometers */  
  
    /* Get the distance in miles. */  
    printf("Enter the distance in miles> ");  
    scanf("%lf", &miles);  
  
    /* Convert the distance to kilometers. */  
    kms = KMS_PER_MILE * miles;  
  
    /* Display the distance in kilometers. */  
    printf("That equals %f kilometers.\n", kms);  
  
    return (0);  
}
```

direttive di preprocessore

commenti

programma principale

parole riservate



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa



# Parole riservate

- Sono le *parole* del linguaggio C

- Non possono essere usate dall'utente

- Le parole riservate e quelle definite dall'utente si chiamano *identificatori*

## ANSI C RESERVED WORDS

auto  
break  
case  
char  
const  
continue  
default  
do

double  
else  
enum  
extern  
float  
for  
goto  
if

int  
long  
register  
return  
short  
signed  
sizeof  
static

struct  
switch  
typedef  
union  
unsigned  
void  
volatile  
while



Università  
degli Studi  
di Palermo

dj  
dipartimento  
di ingegneria  
unipa



# Identificatori

- I nomi che possono essere scritti dall'utente, oltre che le parole riservate
  - Variabili, funzioni, nomi di macro ...
- Regole di formazione di un identificatore definito dall'utente:
  1. Consiste solo di lettere, cifre e il carattere \_ (underscore)
  2. Non può iniziare con una cifra
  3. Non può essere una parola riservata
  4. Gli identificatori definiti nella libreria standard non possono essere riutilizzati

# Identificatori

```
letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable
```

```
1Letter  
double  
int  
TWO*FOUR  
joe's
```

Validi o non validi?



Università  
degli Studi  
di Palermo

dipartimento  
di ingegneria  
unipa



# Identificatori

`letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable`

`1Letter`

begins with a letter

`double`

reserved word

`int`

reserved word

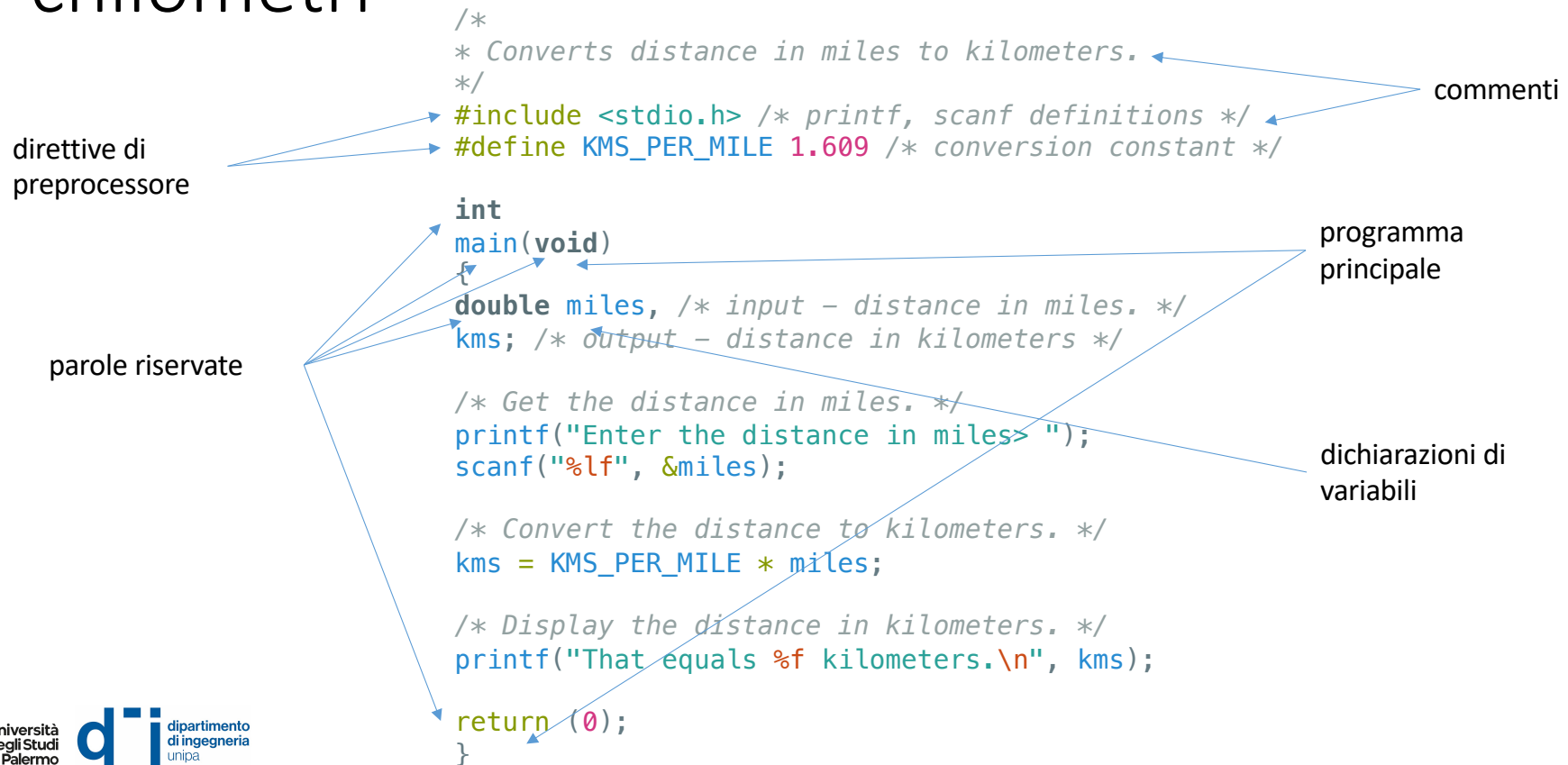
`TWO*FOUR`

character `*` not allowed

`joe's`

character `'` not allowed

# Un programma per convertire le miglia in chilometri





# Dichiarazioni di variabili

- Riserviamo spazio in memoria per l'informazione riferita dalla variabile
- Le singole variabili anche dello stesso tipo possono essere dichiarate separatamente
- Le variabili possono essere anche inizializzate all'atto della dichiarazione

```
SYNTAX:      int variable_list;
              double variable_list;
              char variable_list;

EXAMPLES:    int count,
              large;
              double x, y, z;
              char first_initial;
              char ans;
```

```
int a = 3, b;
double x = 4.0, y = 3.5,
       z = -1.324e-10;
```

# Tipi di dato e letterali

- I diversi tipi di dato (`char`, `int`, `float`, `double`, ...) corrispondono ad allocare quantità di memoria diverse
- *Letterale*:
  - La forma sintatticamente corretta per esprimere i valori costanti di un dato tipo

# Tipi di dato e letterali

- `int`, `short`, `long`  
...

- Nasce come rappresentazione a 16 bit dei numeri interi con segno

-10500      435      +15      -25      32767

`short`

-32,767 .. 32,767

`unsigned short`

0 .. 65,535

`int`

-2,147,483,647 .. 2,147,483,647

- Esistono *tipi interi* di diverse lunghezze, con e senza segno

`unsigned`

0 .. 4,294,967,295

`long`

-2,147,483,647 .. 2,147,483,647

`unsigned long`

0 .. 4,294,967,295

# Tipi di dato e letterali

- float, double,  
long double

## Valid double Constants

---

- Rappresentazione dei  
reali in virgola mobile  
a *diverse precisioni*

3.14159

0.005

12345.0

15.0e-04 (value is 0.0015)

2.345e2 (value is 234.5)

1.15e-3 (value is 0.00115)

12e+5 (value is 1200000.0)



Università  
degli Studi  
di Palermo

**dj**  
dipartimento  
di ingegneria  
unipa



# Tipi di dato e letterali

- float, double, long double

- Rappresentazione dei reali in virgola mobile a *diverse precisioni*

## Valid double Constants

3.14159  
0.005  
12345.0  
15.0e-04 (value is 0.0015)  
2.345e2 (value is 234.5)  
1.15e-3 (value is 0.00115)  
12e+5 (value is 1200000.0)

## Invalid double Constants

150 (no decimal point)  
.12345e (missing exponent)  
15e-0.3 (0.3 is invalid exponent)  
12.5e.3 (.3 is invalid exponent)  
34,500.99 (comma is not allowed)

# Tipi di dato e letterali

- float, double, long double
- Rappresentazione dei reali in virgola mobile a *diverse precisioni*

Type	Approximate Range*	Significant Digits*
float	$10^{-37} \dots 10^{38}$	6
double	$10^{-307} \dots 10^{308}$	15
long double	$10^{-4931} \dots 10^{4932}$	19

In una *tipica implementazione* di C

# Tipi di dato e letterali

- char
- Rappresenta singoli caratteri, cifre, segni di interpunzione, *caratteri non stampabili*

'A' 'z' '2' '9' '\*' ':' '"' ' '

# Tipi di dato e letterali

- `char`
- Rappresenta singoli caratteri, cifre, segni di interpunzione, *caratteri non stampabili*
- Vengono rappresentati tramite le tabelle dei caratteri
  - ANSI C usa il *codice ASCII*

Right Digit		ASCII								
Left Digit(s)	0	1	2	3	4	5	6	7	8	9
3			□	!	"	#	\$	%	&	'
4	(	)	*	+	,	−	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	/	]	^	−	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}				



# Tipi di dato e letterali

- char
- Rappresenta singoli caratteri, cifre, segni di interpunzione, *caratteri non stampabili*
- Vengono rappresentati tramite le tabelle dei caratteri
  - ANSI C usa il *codice ASCII*

Right Digit		ASCII									
Left Digit(s)	0	1	2	3	4	5	6	7	8	9	
3			□	!	"	#	\$	%	&	'	
4	(	)	*	+	,	-	.	/	0	1	
5	2	3	4	5	6	7	8	9	:	;	
6	<	=	>	?	@	A	B	C	D	E	
7	F	G	H	I	J	K	L	M	N	O	
8	P	Q	R	S	T	U	V	W	X	Y	
9	Z	[	/	]	^	-	`	a	b	c	
10	d	e	f	g	h	i	j	k	l	m	
11	n	o	p	q	r	s	t	u	v	w	
12	x	y	z	{		}					

# Tipi di dato e letterali

- `char`
- Rappresenta singoli caratteri, cifre, segni di interpunzione, *caratteri non stampabili*
- Vengono rappresentati tramite le tabelle dei caratteri
  - ANSI C usa il *codice ASCII*

Right Digit		ASCII								
Left Digit(s)	0	1	2	3	4	5	6	7	8	9
3			□	!	"	#	\$	%	&	'
4	(	)	*	+	,	−	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	/	]	^	−	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}				

I caratteri sono *piccoli interi a 8 bit* ordinati: `'0' < '5' < 'A' < 'a'`

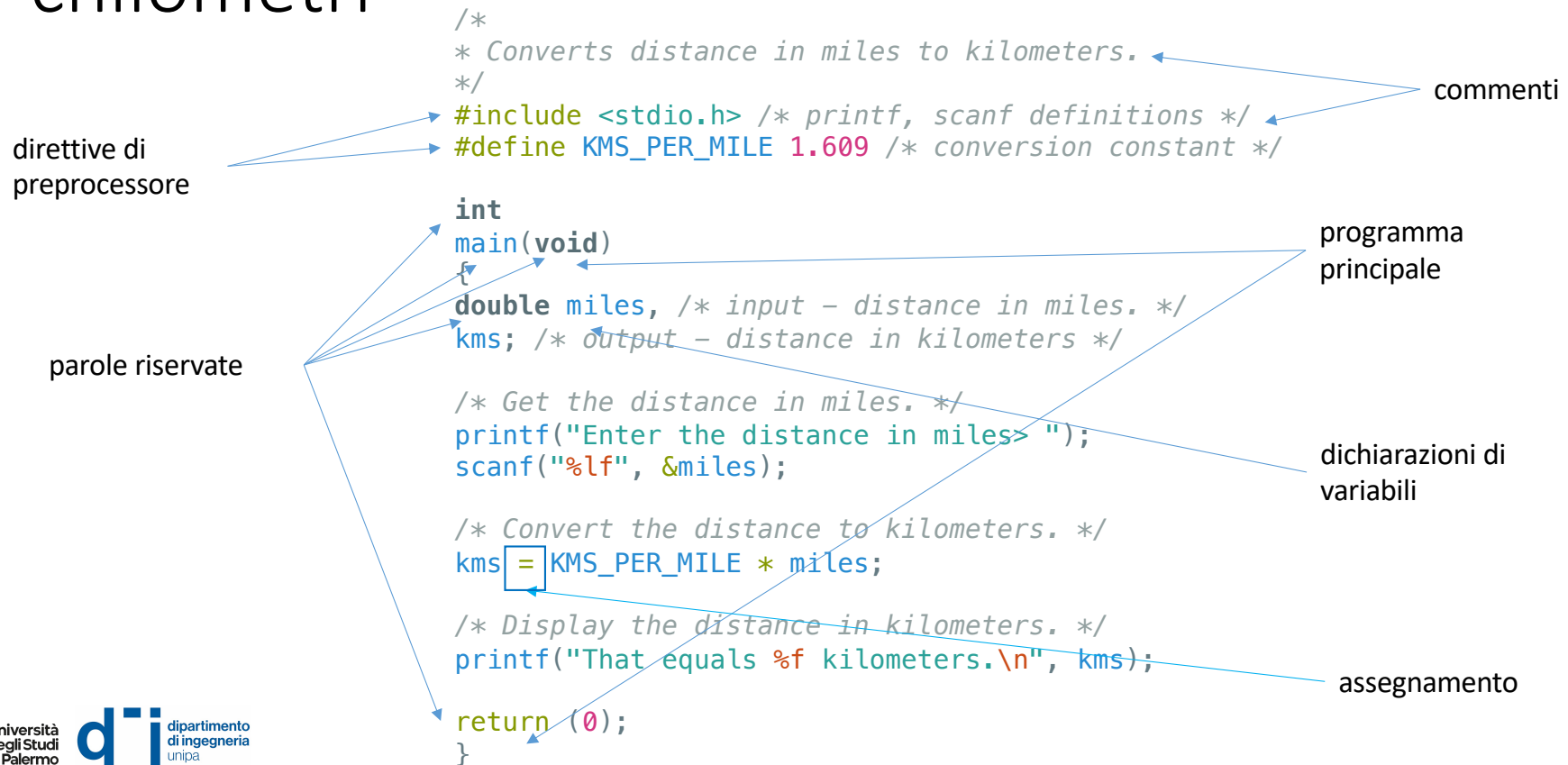
# Tipi di dato e letterali

- `char`
- Rappresenta singoli caratteri, cifre, segni di interpunzione, *caratteri non stampabili*
- Vengono rappresentati tramite le tabelle dei caratteri
  - ANSI C usa il *codice ASCII*

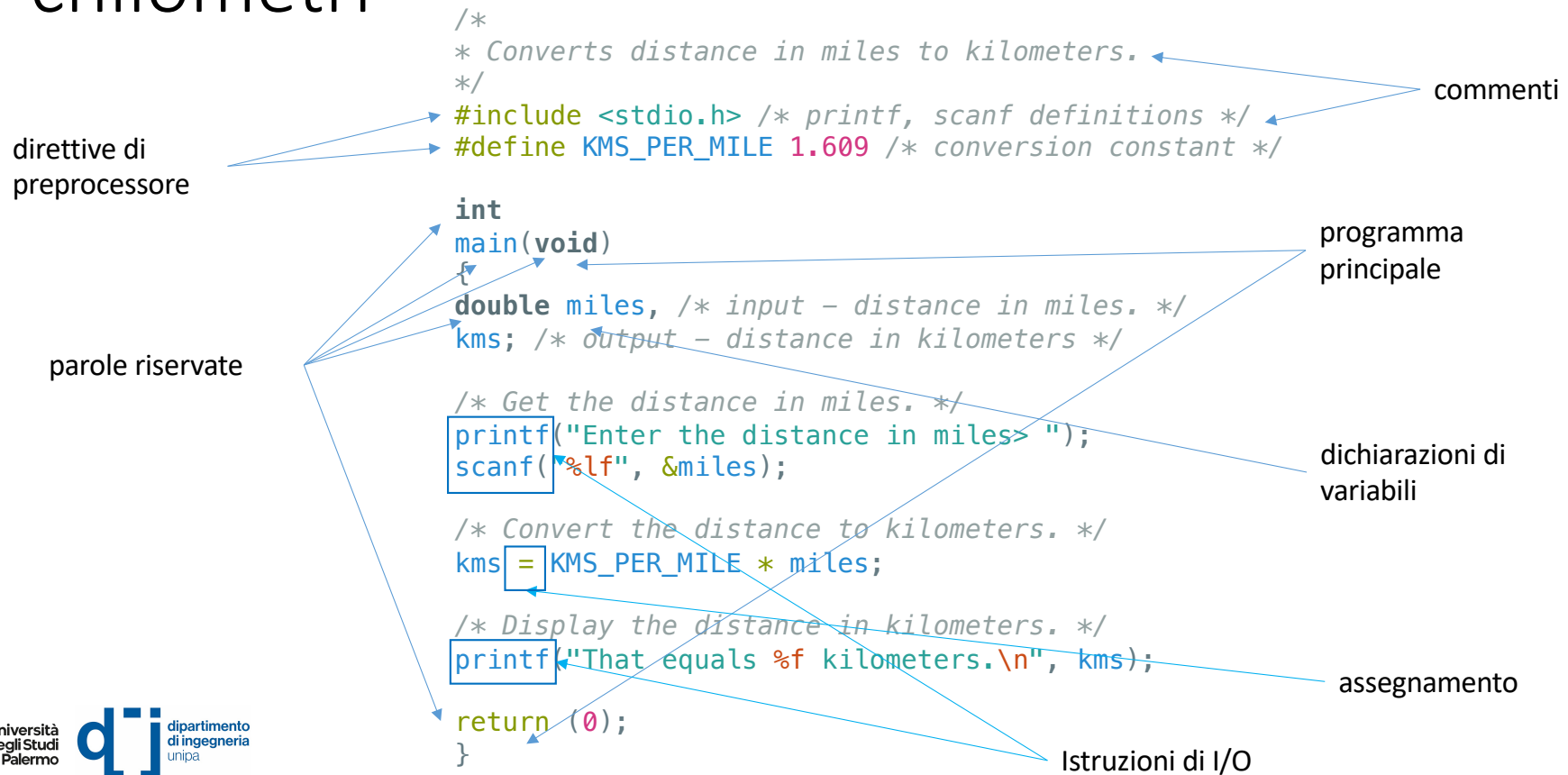
Right Digit		ASCII								
Left Digit(s)	0	1	2	3	4	5	6	7	8	9
3			□	!	"	#	\$	%	&	'
4	(	)	*	+	,	−	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	/	]	^	−	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}				

In una *tipica implementazione* di C `char < short <= int <= long`

# Un programma per convertire le miglia in chilometri



# Un programma per convertire le miglia in chilometri



# Assegnamento

- Usa il simbolo =

FORM: *variable = expression;*  
EXAMPLE: **x = y + z + 2.0;**

- Calcola *expression* e ne inserisce il risultato in *variable*

# Assegnamento

- Usa il simbolo =

FORM: *variable = expression;*  
EXAMPLE: **x = y + z + 2.0;**

- Calcola *expression* e ne inserisce il risultato in *variable*

Before assignment

KMS\_PER\_MILE

1.609

miles

10.00

kms

?

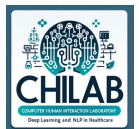
```
kms = KMS_PER_MILE * miles;
```



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa



# Assegnamento

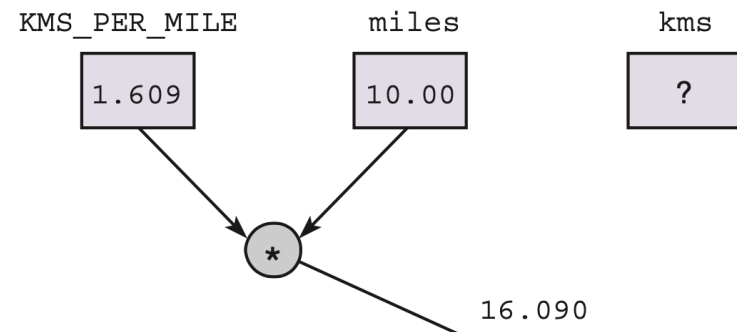
- Usa il simbolo =

FORM:  $variable = expression;$   
EXAMPLE:  $x = y + z + 2.0;$

- Calcola *expression* e ne inserisce il risultato in *variable*

```
kms = KMS_PER_MILE * miles;
```

Before assignment





# Assegnamento

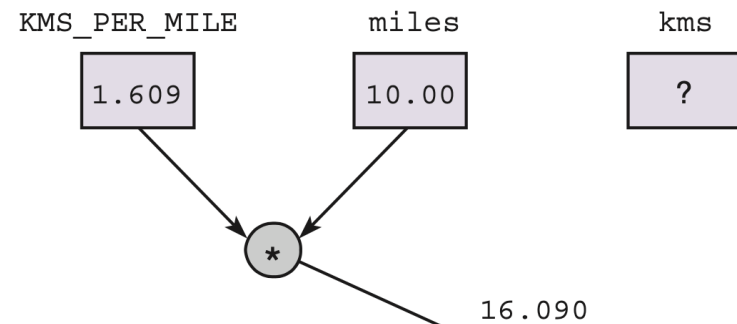
- Usa il simbolo =

FORM:  $variable = expression;$   
EXAMPLE:  $x = y + z + 2.0;$

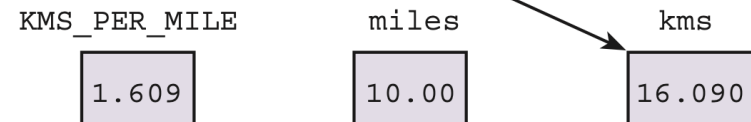
- Calcola *expression* e ne inserisce il risultato in *variable*

```
kms = KMS_PER_MILE * miles;
```

Before assignment



After assignment



# Assegnamento

- Usa il simbolo =

FORM: *variable = expression;*  
EXAMPLE: `x = y + z + 2.0;`

- Calcola *expression* e ne inserisce il risultato in *variable*

Before assignment

sum

100

item

10

```
sum = sum + item;
```



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa



# Assegnamento

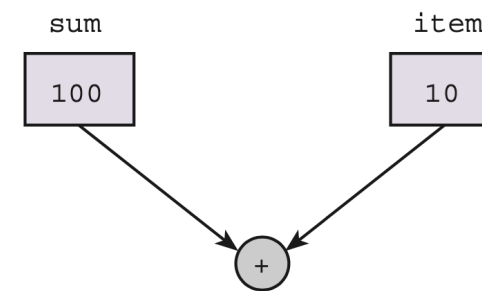
- Usa il simbolo =

FORM:  $variable = expression;$   
EXAMPLE:  $x = y + z + 2.0;$

- Calcola *expression* e ne inserisce il risultato in *variable*

`sum = sum + item;`

Before assignment



# Assegnamento

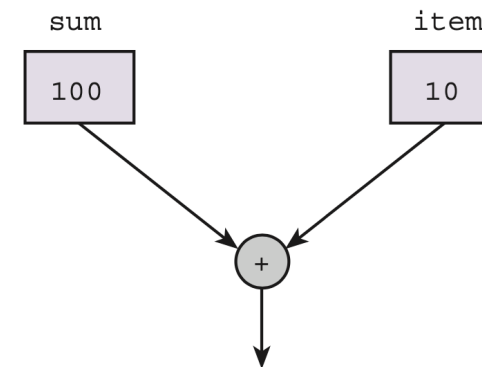
- Usa il simbolo =

FORM:  $variable = expression;$   
EXAMPLE:  $x = y + z + 2.0;$

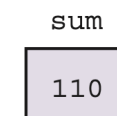
- Calcola *expression* e ne inserisce il risultato in *variable*

`sum = sum + item;`

Before assignment



After assignment



# Istruzioni di I/O

- `printf`

- Stampa a video la *stringa formata* in cui vengono sostituiti i *segnaposto*

SYNTAX: `printf(format string, print list);`

`printf(format string);`

EXAMPLES: `printf("I am %d years old, and my gpa is %f\n",  
age, gpa);`  
`printf("Enter the object mass in grams> ");`

# Istruzioni di I/O

- `scanf`

- Legge da terminale gli elementi di input

SYNTAX: `scanf (format string, input list);`

EXAMPLE: `scanf ("%c%d", &first_initial, &age);`

secondo la *stringa formato* che indica la conversione di tipo da eseguire partendo dal flusso di caratteri digitati

- Necessita di conoscere l'indirizzo di memoria associato alla variabile usando l'operatore indirizzo `&`

# Stringa formato

$\%$ *[allineamento][ampiezza\_campo][.precisione][modificatore] carattere\_tipo*

# Stringa formato

- Tipi e modificatori

Codice	Tipo	Usato da
%d <b>ovvero</b> %i	int	printf /scanf ( <b>solo</b> %d)
%u	unsigned int	printf / scanf
%o	unsigned int ( <b>ottale</b> )	printf / scanf
%x <b>ovvero</b> %X	unsigned int ( <b>esadecimale</b> )	printf / scanf
%h(d, i, u, o, x, X)	<b>tipo intero</b> short	printf / scanf
%l(d, i, u, o, x, X)	<b>tipo intero</b> long	printf / scanf
%c	char	printf / scanf
%f	float	printf / scanf
%e <b>ovvero</b> %E	<b>notazione scientifica</b> (float)	printf / scanf
%g <b>ovvero</b> %G	<b>il minimo tra</b> %e (%E) <b>e</b> %f	printf / scanf
%lf, %lg, %lG	double	printf / scanf
%Lf, %Lg, %LG	double	printf / scanf
%s	<b>stringhe di caratteri</b>	printf / scanf



Università  
degli Studi  
di Palermo



dipartimento  
di ingegneria  
unipa





# Stringa formato

- Ampiezza del campo

- Interi

- `%nd`

- *n* indica lo spazio in caratteri

Value	Format	Displayed Output	Value	Format	Displayed Output
234	<code>%4d</code>	■234	-234	<code>%4d</code>	-234
234	<code>%5d</code>	■■234	-234	<code>%5d</code>	■-234
234	<code>%6d</code>	■■■234	-234	<code>%6d</code>	■■-234
234	<code>%1d</code>	234	-234	<code>%2d</code>	-234

# Stringa formato

- Ampiezza del campo

- Reali

- $\%n.mf$ ,  $\%n.mg$

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	$\%5.2f$	■3.14	3.14159	$\%4.2f$	3.14
3.14159	$\%3.2f$	3.14	3.14159	$\%5.1f$	■3.1
3.14159	$\%5.3f$	3.142	3.14159	$\%8.5f$	■3.14159
.1234	$\%4.2f$	0.12	-.006	$\%4.2f$	-0.01
-.006	$\%8.3f$	■-0.006	-.006	$\%8.5f$	-0.00600
-.006	$\%.3f$	-0.006	-3.14159	$\%.4f$	-3.1416

- $n$  indica lo spazio in caratteri
- $m$  indica il numero di cifre dopo la virgola

# Stringa formato

- Ampiezza del campo
- Stringhe e caratteri

- `%ns`, `%nC`

```
printf("%20s\n", "mamma");
```

- *n* indica lo spazio in caratteri con allineamento a destra

# Stringa formato

- Flag di allineamento

Flag	Significato
-	Allineamento a sinistra
+	Forza il segno nei numeri (anche positivi)
0	padding a sinistra con 0
#	Inserisce la notazione <code>o</code> , <code>0x</code> ovvero <code>0X</code> in esadecimali e ottali
' '	padding a sinistra con spazi

# Espressioni aritmetiche

- Le espressioni aritmetiche, in genere si scrivono come quelle cui siamo abituati dalla matematica
- Usano degli operatori e diversi livelli di parentesi
- Esistono precise regole che gestiscono il calcolo di una espressione complessa
  - *Precedenza* → quale operatore viene considerato per primo
  - *Associatività* → in quale direzione vengono considerati gli operandi

# Espressioni aritmetiche

- Associatività
- Esistono *operatori binari* e *operatori unari*
- $x + y$ :  $+$  è binario e associativo da sinistra
- $-y$ :  $-$  è unario e associativo da destra

# Espressioni aritmetiche

- Precedenza e associatività degli operatori aritmetici in C

Simbolo	Significato	Precedenza	Associatività
( )	parentesi – altera la precedenza	1	-
+ -	segno dell'espressione		destra
* /	moltiplicazione e divisione	2	sinistra
%	resto della divisione		sinistra
+ -	somma e sottrazione (binari)	3	sinistra

In C non esiste l'operatore di elevazione a potenza

# Espressioni aritmetiche

- / e %
- Per i tipi interi calcolano la divisione intera e il resto secondo la formula

$$m = (m/n)*n + m \% n$$



# Espressioni aritmetiche

- / e %

$$3 / 15 = 0$$

$$15 / 3 = 5$$

$$16 / 3 = 5$$

$$17 / 3 = 5$$

$$18 / 3 = 6$$

$$16 / -3 \text{ varies}$$

$$0 / 4 = 0$$

$$4 / 0 \text{ is undefined}$$

$$3 \% 5 = 3$$

$$4 \% 5 = 4$$

$$5 \% 5 = 0$$

$$6 \% 5 = 1$$

$$7 \% 5 = 2$$

$$8 \% 5 = 3$$

$$5 \% 3 = 2$$

$$5 \% 4 = 1$$

$$15 \% 5 = 0$$

$$15 \% 6 = 3$$

$$15 \% -7 \text{ varies}$$

$$15 \% 0 \text{ is undefined}$$

Per i tipi reali / fornisce risultato reale e % non è definito

# Espressioni aritmetiche

- Espressioni con variabili di diversi tipi
- L'espressione assume il tipo più «grande» in bit e necessita di una variabile risultato di quel tipo
- Se così non è, i bit del risultato vengono troncati per entrare nella dimensione del tipo del risultato

# Espressioni aritmetiche

- Espressioni con variabili di diversi tipi
- È possibile forzare una espressione ad assumere il tipo desiderato con l'operatore di *cast*

*(desired\_type) expression*

```
n = (int)(9 * 0.5);
```

# Introduzione alla stringhe

- La maniera più semplice di dichiarare una stringa è come *vettore di caratteri*

```
char string_var[30];  
char str[20] = "Initial value";
```

- Viene allocato lo spazio in memoria per tanti `char` quanto è il valore tra [ e ]
- Come faccio a gestire stringhe di lunghezza variabile?
- Vengono stampati o caricati per forza tutti i caratteri?

# Introduzione alla stringhe

- La stringa inizializzata viene terminata con il *carattere nullo* che si rappresenta come `'\0'` e ha codice ASCII pari a 0

```
char str[20] = "Initial value";
```

[0]		[4]		[9]		[14]		[19]											
I	n	i	t	i	a	l		v	a	l	u	e	\0	?	?	?	?	?	?

- I caratteri successivi a `'\0'` vengono ignorati e non vengono stampati

```
char str[7] = "";
```

Cosa c'è nella stringa?

# Introduzione alla stringhe

- Comportamento con `scanf`
- Non è necessario il carattere `&`
  - In quanto vettore di caratteri, il nome della stringa *è già un riferimento ad una locazione di memoria*

```
char str[5]="";  
printf("> ");  
scanf("%9s",str);  
printf("%s\n",str);
```

Che succede?

# Introduzione alla stringhe

- Comportamento con `scanf`
- Non è necessario il carattere `&`
- In quanto vettore di caratteri, il nome della stringa  
*è già un riferimento ad una locazione di memoria*

```
char str[7]="";  
char str1[7]="";
```

```
printf("> ");  
scanf("%s,%s",str,str1);  
printf("%s\n%s\n",str,str1);
```

Che succede?